Smartphone & Cross-platform Communication Toolkit User Manual for Android

Release 3.0.0

July 2013 Edition

Worldwide technical support and product information:

www.toolsforsmartminds.com

TOOLS for SMART MINDS Corporate headquarter

Via Padania, 16 Castel Mella 25030 Brescia (Italy)

Copyright © 2010 Tools for Smart Minds. All rights reserved.

CONTENTS

CONTENTS **4**

ABOUT THIS MANUAL **6**

CONVENTIONS 6

INTRODUCTION **7**

OVERVIEW 7

TOP REASONS TO USE SMARTPHONE & CROSS PLATFORM COMMUNICATION TOOLKIT (SCCT) 7

SCCT APPLICATION EXAMPLES 8

COMMUNICATING DATA ACROSS A NETWORK, ON A COMPLEX ASSEMBLY LINE 8

SUPPORTED PLATFORMS 9

PUBLISHER LIBRARY 9

SUBSCRIBER LIBRARY 9

INSTALL SCCT SUBSCRIBER ON NETBEANS **ERRORE. IL SEGNALIBRO NON È DEFINITO.**

INSTALL SCCT SUBSCRIBER ON ECLIPSE 10

HOW SCCT FOR ANDROID WORKS **11**

HOW DATA PACKAGES ARE MANAGED **12**

PACKAGES **13**

CONFIGURATION PACKAGE 13

Channel Configuration *13*

Line Configuration *13*

ERROR PACKAGE 14

ANALOG DATA PACKAGE 14

DIGITAL DATA PACKAGE 15

MESSAGE PACKAGE 15

XML PACKAGE 15

FILE PACKAGE 16

Saving files *16*

IMAGE PACKAGE 17

LOCATION PACKAGE 17

ARRAYS 18

FILTERS 19

Array Filter *19*

Analog Data Filter *20*

OPENING AND CLOSING COMMUNICATION **21**

STARTING AND STOPPING DATA TRANSMISSION **22**

RECEIVING DATA **22**

REGISTERING AND UNREGISTERING OBSERVER **23**

SENDING DATA **23**

SOURCES **24**

PERMISSIONS **25**

QUICK START **26**

FIGURE INDEX **28**

About this Manual

Conventions

The following conventions appear in this manual:

▸	The ▸ symbol leads you through nested menu items and dialog box options to a final action. The sequence **Tools** ▸ **Options** directs you to pull down the **Tools** menu, select **Options** item.
Bold	Bold text denotes items that you must select or click on the software, such as menu items and dialog box options. Bold text also denotes parameter names.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excepts.
`monospace italic`	Italic text in this font denotes text that is a placeholder for a word or value that you must supply.

INTRODUCTION

This chapter describes the installation procedure, installed components, and the main features of the Smartphone & Cross-platform Communication Toolkit.

OVERVIEW

The Smartphone & Cross-platform Communication Toolkit is an add-on package for communicating data trough applications. The toolkit contains a set of high level functions for sending your application data and advanced functions for customized tasks.

The following list describes the main features of the Smartphone & Cross-platform Communication Toolkit.

- Works over any TCP/IP connection
- Works over Local Area Networks as well as Internet connections.
- Implements the publisher – subscriber pattern (also known as Observer pattern)
- Authenticates subscribers through an API-KEY.
- Controls in background the state of every connection to identify loss of communication.
- Publishes GPS coordinates to manage mobile systems.
- Works with platform independent Data format and communicate with multiple platforms at the same time.

TOP REASONS TO USE SMARTPHONE & CROSS PLATFORM COMMUNICATION TOOLKIT (SCCT)

Adopting this toolkit you have the following advantages:

Simplify communication: don't care about communication details over a TCP communication channel, SCCT does it for you.

Multiple platforms are supported: exchange your data with a protocol supported on a wide range of platforms and programming languages.

It's reliable: many applications have been created with this toolkit around the world.

Speed up your development activity: this toolkit allows the creation of distributed application and let you save a lot of your time.

Each platform SCCT supports is provided with a library written in its own native language. This guarantees you maximum performances and the possibility to use all features available on any device and operative system.

Today applications need to retrieve information from database, bar code readers, OCR systems, remote data acquisition or technical operators with mobile devices. Often you have to create systems capable

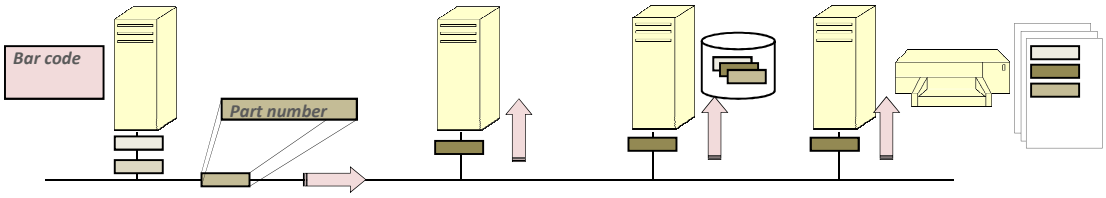
to exchange data with legacy applications created on different platforms. Every time you have to design an ad hoc communication protocol, code both server and client side routines, debug them. Every time! With SCCT you have a new tool that let you save a lot of time, lets you focus on your project's core and manage multiple communications at the time. In the following examples you are going to see how SCCT can dramatically improve the quality of your software solutions.

SCCT APPLICATION EXAMPLES

SCCT can be successfully applied in many real world situations. In this chapter we discover where SCCT helps developers to succeed to deliver high valued solutions.

COMMUNICATING DATA ACROSS A NETWORK, ON A COMPLEX ASSEMBLY LINE

Creating an application that broadcasts part numbers and barcodes coming from databases or bar code readers, to all computers on an assembly line, as schematized below. On modern assembly lines, many computers control single processing stations and they need to exchange data among them to know, part numbers, print codes and certificates, store test results on different databases. SCCT provides many functionality to exchange data with applications developed with heterogeneous programming languages. Besides, when your customer asks you to show some data on a mobile device, SCCT is capable to communicate with a large variety of smart phones and tables and you do not have to change a single line of code of your software. SCCT cares of all connected devices and communicate your data to all of them at the same time, receives user's requests and organize them in a time-ordered FIFO so that you can process them easily.



SUPPORTED PLATFORMS

SCCT is composed by two main components:

- publisher library
- subscriber library

PUBLISHER LIBRARY

This library lets you create a full-featured publisher, which authenticates incoming subscribers, checks connection status, sends data to all active publishers and passes their request to your application.

This library is available as a set of Vis for LabVIEW 2010 or later.

To get more details or download an evaluation copy of this library please visit:

http://www.toolsforsmartminds.com/products/SCCT.php

SUBSCRIBER LIBRARY

This library let you create a subscriber which handles all communication details with a publisher so you don't have to. It receives data packages and present them to your application according to their data types.

This library is available for the following platforms and languages:

Name	***Operating System***	***Development Language***
SCCT Subscriber for LabVIEW	Windows	LabVIEW 2010
SCCT Subscriber for Java	Java VM 5.0 or later	Java
SCCT Subscriber for Android	Android 2.1 or later	Java
SCCT Subscriber for Linux	Linux Kernel 2.6.30	ANSI – C
SCCT Subscriber for iPhone/iPad	iOS	Objective C
SCCT Subscriber for Javascript	HTML5 browser	Javascript

To get more details or download your **free** copy of SCCT subscriber library, please visit:

http://www.toolsforsmartminds.com/products/SCCT.php

INSTALL SCCT SUBSCRIBER ON ECLIPSE

Also in this case you can download a free version of Eclipse from http://www.eclipse.org/downloads/.

In order to use android library, you have also to install the Android SDK. You can find a useful guide on http://developer.android.com/sdk/installing.html .

To add the library on the project you can do:

1. Right click mouse on the project in which you want add it. From drop-down menu click **Properties**
2. Select **Java Build Path i**n the menu on the left

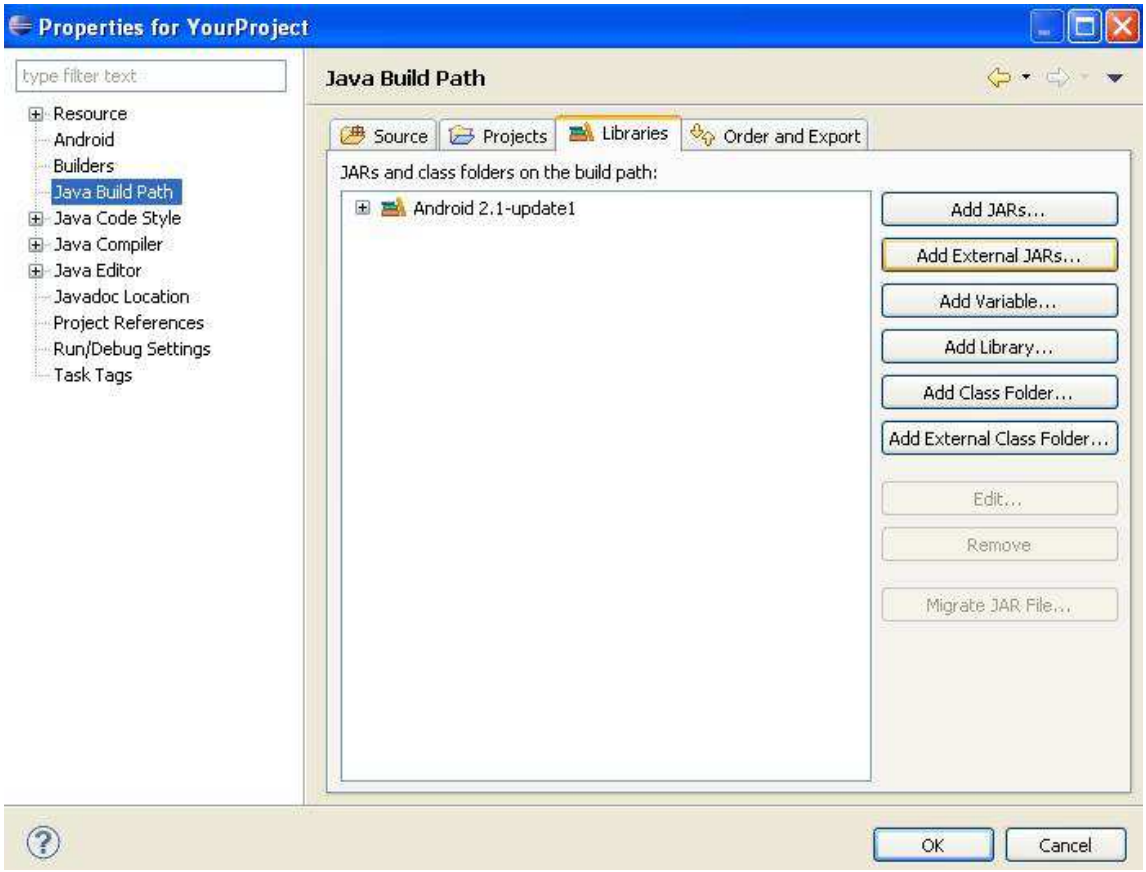


FIGURE 1 - PROJECT PROPERTIES WINDOW ON ECLIPSE

3. Click "**Add External JARs…"** button and select the downloaded version of SCCT subscriber library indicating path.
4. Click **OK** to close the window

How SCCT for Android works

SCCT subscriber for Android is a library included in your app that take care of communication with SCCT publisher for LabVIEW, code/decode packages in appropriate way, and so on.

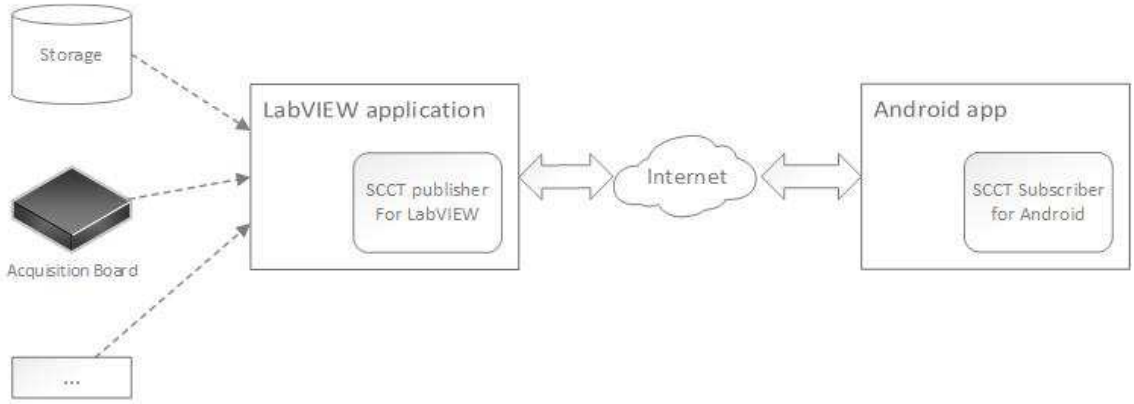


Figure 2 - Communication scheme

Its goal is make life easy for developers. To do this SCCT uses the event-oriented paradigm, implementing the Observer pattern. This is a pattern intuitively used as an architectural base of a lot of event management system.

The pattern is based on two classes: subject and observer. The subject has the role of notify all registered observers whenever there are new events or new data to use. The observer, that is registered to the subject, manages events when this occurs.

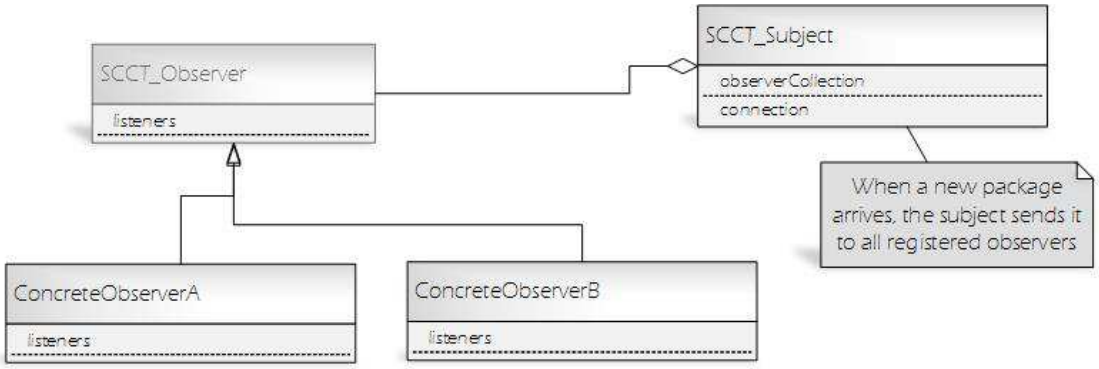


Figure 3 - UML scheme of Observer pattern

Therefore, SCCT Subscriber for Android provides you two fundamental classes for this work:

- `SCCT_Subject`: has the task of handling directly the communication with the publisher, provides some methods to manage the connection and send packages to publisher and provides received data to Observer objects.
- `SCCT_Observer` : provides different kinds of listener to receive data and manage connection events. This listeners have to be overridden by concrete observer subclasses to define how manage this data and events.

`SCCT_Observer` object must register itself to `SCCT_Subject` if it want to receive data. More observer objects can be registered in the same subject.

The developer has to implement one or more concrete observers to define how handles the incoming packages or to define how its app should behave when an events occurs.

How data packages are managed

When a new package arrives, SCCT_Subject processes it and sorts it to all registered observers. Each `SCCT_Observer` puts the received package in its own queue. The same package is shared among the observers and it's accessible only in read-only mode. This approach guarantees the safety and integrity of the information avoiding memory waste.

Packages are sorted in queues by order of arrival as shown below. Each `SCCT_Observer`, then, extracts the first package from the queue and sends it in the appropriate listener. If the listeners are too slow to process the packages, the queue size can grows indefinitely. To avoid this, the developer should avoid to make too long task in the listeners, otherwise he should unregister the observer or stop the communication when the queue grow up too much. To see how much element are in the queue, you can use `getAwaitingPackages()` methods of `SCCT_Observer`.

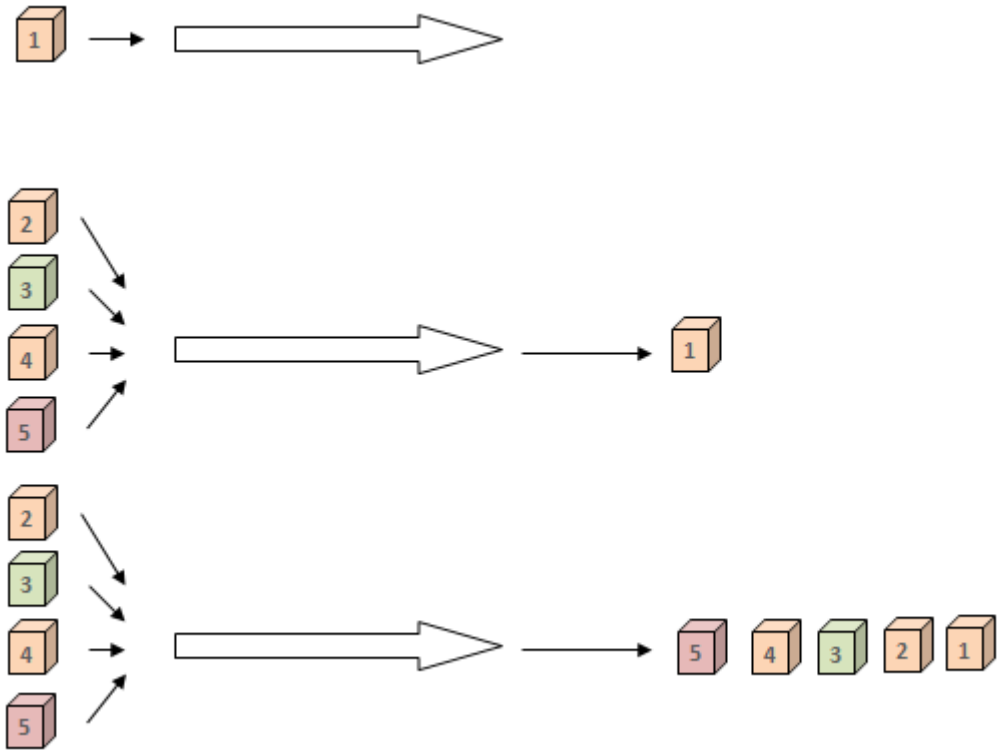


Figure 4 - How packages are sorted in queues

Also the packages awaiting to be transmitted are enqueued in the transmission queue. This allows to send more packages avoiding to wait for them. A progress id is assigned to all sent packages and the `getLastPackagesSent()` method of `SCCT_Subject` returns the id of last transmitted package. You can also see how many packages are still awaiting to be transmitted using the `getAwaitingPackages()` methods of `SCCT_Subject`.

PACKAGES

SCCT manages several types of packages containing different kind of data. These packages are used to exchange data between publisher and subscriber in easy and fast way.

CONFIGURATION PACKAGE

Configurations are contained in the `SCCT_ConfigurationPackage` object that holds information about the used device, like name and type of the device, server location, channels and lines configurations, etc.

Usually this package is sent at the beginning of the transmission to inform the client about how many lines and channels are used and which configuration they have.

CHANNEL CONFIGURATION

To get a `SCCT_ChannelConfiguration` object you have to call `getChannelConfiguration(int` **`index`**`)` method of `SCCT_ConfigurationPackage`, with 0 ≤ **index** < `getNumberOfChannels()`.

This object holds the following data that you can get with respective getters:

- **Description**: the name or the description of the channel;
- **Unit**: the measurement unit of the channel;
- **SamplingRate**: the sampling rate of the channel;
- **MinValue**: the minimum value that the channel can assume;
- **MaxValue**: the maximum value that the channel can assume;
- **NChannel**: the index of the channel.

LINE CONFIGURATION

To get a `SCCT_LineConfiguration` object you have to call `getLineConfiguration(int` **`index`**`)` method of `SCCT_ConfigurationPackage`, with 0 ≤ **index** < `getNumberOfLines()`.

This object holds the following data that you can get with respective getters:

- **Description**: the name or the description of the line;
- **LineNumber**: the index of the line.

ERROR PACKAGE

The errors are generated when a problem occurs while connecting and are managed with an SCCT_`ErrorPackage`. Each error is associated with a code (in the SCCT_`ErrorPackage` class there are some constants to help the developer). **Whenever an error occurs the connection is broken.**

This error can be generated because of one of these reasons:

Constant	Description
ERROR_WRONG_API_KEY	This error is thrown if API-Key is incorrect.
ERROR_WRONG_TIMEOUT	This error is thrown if timeout is incorrect.
ERROR_WRONG_TIMESTAMP	This error is thrown if the client's timestamp is too much different from the producer's one.
ERROR_EXPIRED_TIMEOUT	This error is thrown if timeout is expired.
ERROR_TRANSMISSION_ERROR	This error is thrown if it fails to send a message.
ERROR_LOST_CONNECTION	This error is thrown if SCCT notices that the connection is lost before the timeout expired.
ERROR_DESTINATION_UNREACHABLE	This error is thrown if SCCT is not able to establish a connection with the publisher.

The last four errors are generated by the library and have a different timestamp from other packages (that are created by the producer).

ANALOG DATA PACKAGE

Analog data are the sampled values of analog channels and are contained in the SCCT_`AnalogDataPackage` object. You can take these data with **`public double[][]`** `getData()` method that returns a *n x m* matrix of double: every row represents a channel, and every column represents a datum of that channel.

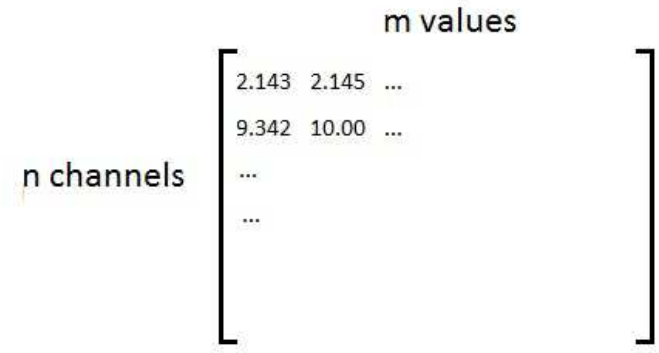


FIGURE 5 - ANALOG DATA MATRIX

You can take the number of channels with **`public int`** `getNumOfChannels()` method and the number of values with **`public int`** `getSamples()` method.

You can use these data to show them in a label, draw them in a graph or store them in a database or in a file.

Digital Data Package

Digital data are values of digital lines and are contained in the SCCT_DigitalDataPackage object. You can take these data with **public boolean**[] getData() method that returns an array of *n* booleans: every value of array represents a digital line.

You can take the number of lines with **public int** getNumberOfLines() method.

Message Package

A SCCT_MessagePackage object contains a short message with an associate code number. This kind of package is useful for exchange messages or commands with the publisher (e.g. to request some data or to tell it to start a task).

The **public** String getMessage() method returns the message received, **public int** getMessageCode() method instead returns the code associated.

Xml Package

An SCCT_XmlPackage object contains an xml document. This class doesn't validate the xml, so it can be also contained a malformed xml: is a developer responsibility to validate the document if necessary. To get the xml document you have to call **public** String getCustomXml() method that returns a String object. This kind of package is similar to message package, but it has a different semantic purpose and then is handled differently from SCCT.

FILE PACKAGE

SCCT allows you to exchange easily any kind of file between publisher and subscriber through the SCCT_`FilePackage` class.

This class holds the following data:

- **File name:** The file name. You can get it with **`public`** `String getFileName()` method;
- **File:** This is a byte array containing the file received or sent. You can get it with **`public byte`**`[] getBytes()` method;
- **Md5:** the md5 code of file. This field is optional, so if the publisher sent the file without calculating md5 code, this String will be empty. When you send a file you can decide if calculate it with a boolean value in the constructor. You can get it with **`public`** `String getMd5()` method;
- **Attributes:** a list of file attributes (e.g. the file author name, the creation date etc.). You can get it through **`public`** `String[] getAttributes()` method. This field is optional, so if the publisher sends the file without it, the array will be empty.

NB: This package is available only in PRO version.

SAVING FILES

When a SCCT_`FilePackage` arrives you can take the file and manipulate its body with **`public byte`**`[] getBytes()` method, but often can be enough for you saving it on the disk: for this reason SCCT provides you some methods to save file in an easy way.

There are two main ways for saving a file:

- The first one involves file saving associated with Context application package. In order to use it you have to call the following method:

```
public void saveFile(String path, Context context, int mode) throws IOException
```

This method have the following parameter:

- **Path:** The path where it saves the file (e.g. if the value of this string is "test/" the file will saved in "test" subdirectory). By default this parameter is an empty string;
- **Context:** Is the Context of application used to save the file;
- **Mode:** It's the operation Mode. This value can be Context.MODE_PRIVATE (Used by default), Context.MODE_APPEND, Context.MODE_WORLD_READABLE or Context.MODE_WORLD_WRITEABLE.

If an error occurs in an attempt to save the file, an IOException is thrown.

- The second one consists in saving it on external storage. In order to use it you have to call the following method:

```
public File saveFileOnExternalStorage(String path,boolean append)throws IOException
```

This method requires android.permission.WRITE_EXTERNAL_STORAGE permission to be used, as explained in "Permission" chapter.

This method has the following parameters:

- **Path:** path where it saves the file (e.g. if the value of this string is "test/" the file will saved in "test" subdirectory). By default this parameter is an empty string;
- **Append:** If true, then bytes will be written to the end of the file rather than the beginning. (By default is false).

If an error occurs in an attempt to save the file, an IOException is thrown.

IMAGE PACKAGE

SCCT allows you to exchange easily images between publisher and subscriber through the `SCCT_ImagePackage` class.

This class holds the following data:

- **Description:** The description or the name of the image. You can get it with **`public`** `String getDescription()` method;
- **Image:** This is the image received or sent. You can get it with **`public byte`**`[] getBytes()` method to obtain a byte array containing the image. You can also get a `Bitmap` object representing the image with **`public`** `Bitmap getBitmap()` method[1] ;
- **Attributes:** a list of file attributes (e.g. the image author name, the creation date etc.). You can get it through **`public`** `String[] getAttributes()` method. This field is optional, so if the publisher sends the file without it, the array will be empty;
- **Format:** It's the format of the image. You can take it with **`public int`** `getImageFormat()` method. This field can take one of the following constants value:
  - `SCCT_ImagePackage.FORMAT_PNG;`
  - `SCCT_ImagePackage.FORMAT_JPEG;`
  - `SCCT_ImagePackage.FORMAT_BMP;`
  - `SCCT_ImagePackage.FORMAT_TIFF`.

NB: This package is available only in PRO version.

LOCATION PACKAGE

A `SCCT_LocationPackage` package holds the GPS coordinates sent by the server. This package can indicate the location of the server or of a specific sensor, for example. The information that it provides are the latitude, the longitude, the elevation and a description associated with it. You can access these information with relative getter methods.

NB: This package is available only in PRO version.

[1] This method works only with supported Android codec, otherwise returns a blank image (see http://developer.android.com/guide/appendix/media-formats.html for more information).

ARRAYS

The array packages, that inherit from the SCCT_2DArrayPackage abstract class, are a powerful way to exchange two-dimensional arrays of different primitive types. The supported types are:

Type	**Package**
double	SCCT_2DDoubleArray
float	SCCT_2DFloatArray
integer	SCCT_2DIntegerArray
short	SCCT_2DShortArray
long	SCCT_2DLongArray
boolean	SCCT_2DBoolArray
String	SCCT_2DStringArray

You can get the dimension of the array with the `getRowCount()` and `getColumnCount()` methods. To obtain data stored in the object you have to call `get2DArray()` method.

Each array package has also an associated filter id (obtainable with `getFilterId()` method) that you can use to see which filter is applied on these data. For more information about filters see Filters chapter.

To receive different kinds of array, you have to override the appropriate listeners (one for each type of array) provided by the SCCT_Observer class (see Receiving Data chapter for more information), otherwise you can implement only the `arrayListener(SCCT_2DArrayPackage array)` listener and cast the array according to array type.

For example, if you want to receive a String array, you can use the specific listener:

```
@Override
public  void stringArrayListener(SCCT_2DStringArray stringArray){
	String [][] data = stringArray.get2DArray();
	//Do something with string array
}
```

Otherwise, you can use the generic listener:

```
@Override
public  void arrayListener(SCCT_2DArrayPackage array){
	if(array.getType() == SCCT_ArrayType.STRING){
		SCCT_2DStringArray stringArray = (SCCT_2DStringArray) array;
		String [][] data = stringArray.get2DArray();
		//Do something with string array
	}
}
```

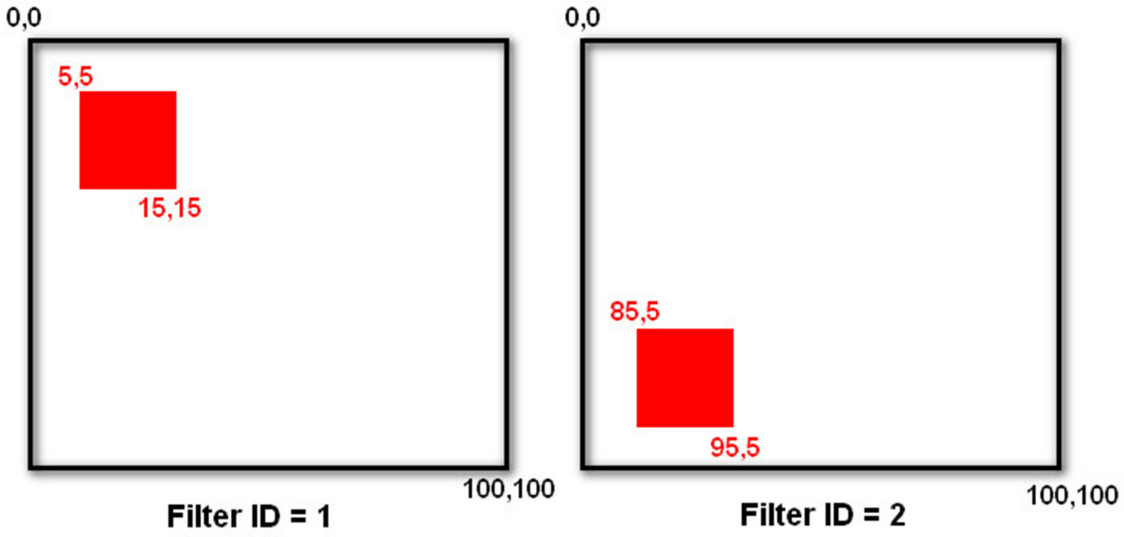
These two solutions are completely equivalent. The only difference between the two approaches is that `arrayListener` accept all kinds of array, so you have to control the incoming array type and cast it appropriately. This can be useful to avoid to implement many listener in same situation.

FILTERS

To reduce the amount of bandwidth used to communicate and the cpu and memory utilization on client device, SCCT 3.0 has introduced filter packages. Filters allow subscribers to request a subset of data transmitted by the publisher.

Filters is available for analog data and for 2D array of boolean, String, int, long, short, double, and float. For analog data, filtering allows to select specific channels. Otherwise, for 2D arrays, filtering permits you to select a subset of the available published array data.

Each instantiated filter is identified with a progress id called `filterId`. The `filterId` permits the client to relate every received data package to the relative filter request. `filterId` is fundamental to process received data in the right way. For example, if server manages a 100x100 array and a client needs a 11x11 subarray (from index 5,5 to 15,15), the client sends a filter request. As it is the first filter request sent to the server, the filter request must generate a `filterId = 1`. When the filter request is performed, server sends the required subarray and marks transmitted packages with `filterId = 1`. Later, if the client needs a different 11x11 subarray, it sends a another filter request. As it is the second filter request, the filter request generates a `filterId = 2`. The server performs this new filter request, sends the new subarray and marks transmitted packages with `filterId = 2`. The following figure schematizes this example. The use of `filterId` permits the client to link every received package with the first or the second filter request.



ARRAY FILTER

Filters of array are represented by `SCCT_ArrayFilter` class. In the constructor you have to indicate the array type on which apply the filter.

To select columns and rows you can use the following methods:

`public void selectColoumns(int from, int to)`	`Select the columns from` *`from to to`*`.`
`public void selectRows(int from, int to)`	`Select the rows from` *`from to to`*`.`
`public void setQuery(String query)`	The query syntax is: "r1-r2;c1-c2" where r1 is the start row, r2 is the end row, c1 is the start column and c2 is the end column.

The following example explains how to select a subarray of double from index 5,5 to 15,15 .

```
SCCT_ArrayFilter filter = new SCCT_ArrayFilter(SCCT_ArrayType.DOUBLE);
 //SCCT_ArrayType indicates the kind of array you want to apply the filter to.
filter.setQuery("5-15;5-15");
/*
        Either way, you can use the following code:
        int from = 5;
        int to = 15;
        filter.selectRows(from,to);
        filter.selectColumns(from,to);
*/
subject.sendFilter(filter);
```

ANALOG DATA FILTER

Analog data filter is represented by `SCCT_AnalogDataFilter` class. In analog data filter you can select the channels (also scattered) and the sources to receive.

To select channels and columns you can use the `following methods:`

`public void selectChannels(int from, int to)`	`Select the channels from from to to.`
`public void selectChannels(int[] channels)`	`Select the indexes of channels in the array.`
`public void selectSourcs(int[] sources)`	`Select the ids of sources in the array.`
`public void setQuery(String query)`	`The query syntax is: "c1-c2//s1,s2" or alternatively "c3,c4//s1,s2". In the first case are selected the channels from c1 to c2. In the second case, instead, the channels c3 and c4 only are selected. In both cases the sources s1 and s2 are selected.`

The following example explains how to select the channels from 1 to 3 of the sources 0 and 2.

```
SCCT_AnalogDataFilter  filter = new SCCT_AnalogDataFilter();
filter.setQuery("1-3//0,2");
/*
        Either way, you can use the following code:
        filter.selectChannels(1,3);
        filter.selectSources(new int []{0,2});
*/
subject.sendFilter(filter);
```

OPENING AND CLOSING COMMUNICATION

`SCCT_Subject` is the class that deals with the communication: each data passes through it. Each object handles a single connection with a publisher, if you want to communicate at the same time with more publishers you have to instantiate more objects. In most cases, however, you needs only a subject at a time. In these cases you can use the **`public static`** `SCCT_Subject getInstance()` method that returns a single instance[2]. This method is useful for example to share easily the subject among different classes or activities.

To open the connection with a publisher you have to invoke the following method:

```
public void openCommunication(String address, int port, String APIKey,
String description, int timeout, boolean sourceFilterEnabled) throws
IOException
```

This method has the following parameters:

- **Address:** The IP address or the host name of the publisher;
- **Port:** The port number of the publisher;
- **ApiKey:** The password for authenticating the client: if it's wrong, you receive an ErrorPackage;
- **Description:** The name or the description of the client, useful for the publisher to identify which client is connected;
- **Timeout:** The timeout value in seconds. If the connection is lost and the timeout expired the library closes the communication and sends an expired timeout error to observers. This value must be bigger than 1. By default this value is 8;
- **SourceFilterEnabled:** This field enable or disable the source filter (see *Source* chapter for more information). By default this value is *false.*[3]

An `IOException` can be thrown for one of the following reasons:

Name	Description
Empty address	This error is thrown if the address string is empty.
Timeout not correct	This error is thrown if the timeout is less than or equal to one.
Wrong port	This error is thrown if the port is less than or equal to zero.
Connection in progress	This error is thrown if subject was already trying to connect with publisher.
A connection is already active	This error is thrown if a connection is already active.

To close the communication, instead, you have to invoke the following method:

```
public boolean closeCommunication()
```

[2] SCCT_Subject implements the Singleton pattern.

[3] This parameter can be modified only in PRO version, in Lite version the sources are not included.

STARTING AND STOPPING DATA TRANSMISSION

When the communication is open a connection between publisher and subscriber is established, but the publisher waits a client command to begin sending data. However, some packages are sent independently of the connection status (e.g. a configuration or a message error).

To start data transmission you have to call **`public int start()`** method, whereas you have to call **`public int stop()`** to stop it.

RECEIVING DATA

To begin receiving data when a connection is open you have to create a *ConcreteObserver* class that extends `SCCT_Observer` class. `SCCT_Observer` provides a public interface to switch the received packages in different listener methods. It's an abstract class, then you can't instantiate it, but you can extend it and override its listener methods with your concrete class.

Following there is a table with a listener associated with each kind of package:

Package	**Listener**
SCCT_ConfigurationPackage	public void configurationListener (SCCT_ConfigurationPackage conf)
SCCT_ErrorPackage	public void errorListener(SCCT_ErrorPackage error)
SCCT_AnalogDataPackage	public void analogDataListener(SCCT_AnalogDataPackage analogData)
SCCT_DigitalDataPackage	public void digitalDataListener (SCCT_DigitalDataPackage digitalData)
SCCT_MessagePackage	public void messageListener(SCCT_MessagePackage message)
SCCT_XmlPackage	public void xmlListener(SCCT_XmlPackage xmlPackage)
SCCT_FilePackage	public void fileListener (SCCT_FilePackage filePackage)
SCCT_ImagePackage	public void imageListener (SCCT_ImagePackage imagePackage)
SCCT_LocationPackage	public void locationListener(SCCT_LocationPackage locationPackage)
SCCT_2DArrayPackage	public void arrayListener(SCCT_2DArrayPackage arrayPackage)
SCCT_2DStringArray	public void stringArrayListener(SCCT_2DStringArray stringArray)
SCCT_2DDoubleArray	public void doubleArrayListener(SCCT_2DDoubleArray doubleArray)
SCCT_2DFloatArray	public void floatArrayListener(SCCT_2DFloatArray floatArray)
SCCT_2DIntegerArray	public void integerArrayListener(SCCT_2DIntegerArray integerArray)
SCCT_2DLongArray	public void longArrayListener(SCCT_2DLongArray longArray)
SCCT_2DShortArray	public void shortArrayListener(SCCT_2DShortArray shortArray)
SCCT_2DBoolArray	public void boolArrayListener(SCCT_2DBoolArray boolArray)

REGISTERING AND UNREGISTERING OBSERVER

After having created the *ConcreteObserver*, it is not able yet to receive the packages; you have to register it to `SCCT_Subject` object with **`public void`** `registerObserver(SCCT_Observer o)` method. From this moment each package that arrives at the subject will be switched in one of the observer listeners.

When you no longer need the observer, you can unregistering it with **`public void`** `unregisterObserver(SCCT_Observer o)` method.

SENDING DATA

To send data to the publisher, you have to use one of the following method of `SCCT_Subject` object depending on the kind of package you want to send:

Package	**Method**
SCCT_MessagePackage	public int sendMessage(String message,int code) public int sendMessage(SCCT_MessagePackage messagePackage)
SCCT_XmlPackage	public int sendXml(String xml) public int sendXml(SCCT_XmlPackage xmlPackage)
SCCT_FilePackage	public int sendFile(SCCT_FilePackage filePackage)
SCCT_ImagePackage	public int sendImage(SCCT_ImagePackage imagePackage)
SCCT_LocationPackage	public int sendLocation(SCCT_LocationPackage locationPackage)
SCCT_2DStringArray	public int sendArray(SCCT_2DArrayPackage arrayPackage)
SCCT_2DDoubleArray	public int sendArray(SCCT_2DArrayPackage arrayPackage)
SCCT_2DFloatArray	public int sendArray(SCCT_2DArrayPackage arrayPackage)
SCCT_2DIntegerArray	public int sendArray(SCCT_2DArrayPackage arrayPackage)
SCCT_2DLongArray	public int sendArray(SCCT_2DArrayPackage arrayPackage)
SCCT_2DShortArray	public int sendArray(SCCT_2DArrayPackage arrayPackage)
SCCT_2DBoolArray	public int sendArray(SCCT_2DArrayPackage arrayPackage)
SCCT_ArrayFilter	public int sendFilter(SCCT_Filter filter)
SCCT_AnalogDataFilter	public int sendFilter(SCCT_Filter filter)

SOURCES

Sometimes data can come from different sources (e.g. the publisher can handle two different device or take data from different databases). SCCT gives each source an ID and a description.

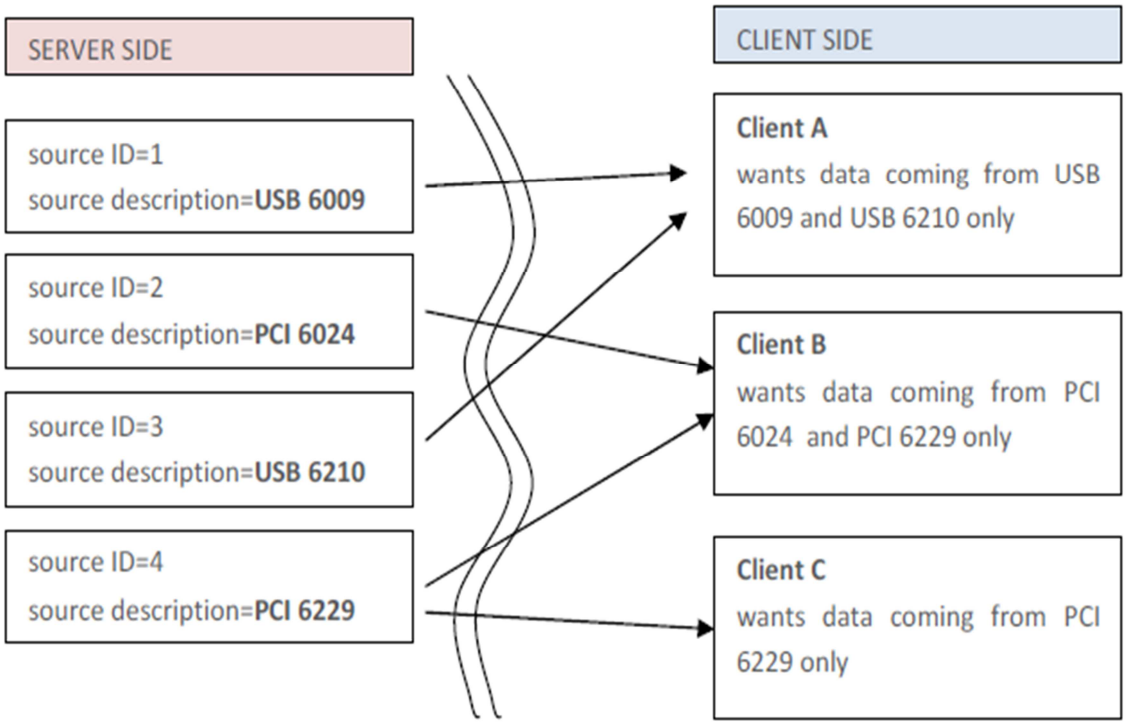


FIGURE 6 - SCENARIO WITH A MULTIPLE DATA SOURCE SERVER AND CLIENTS THAT NEED A SUBSET OF PUBLISHED DATA.

SCCT allows you to operate with different sources in an easy way[4]: if the *"SourceFilterEnabled"* flag in the `openCommunication` method (see *Opening and closing communication* chapter) is enabled the publisher will send only the packages of selected source list. By default this list is empty, but you can modify it with `public boolean setSelectedSourceList(ArrayList<Integer> list)` method, where the list contains the ID of selected sources. If you call again this method the old selected source list is overridden and then the old list is not cumulative with the new one. If, however, *"SourceFilterEnabled"* flag is false the publisher will send the packages of all sources.

To see which source the package arrives from, you can use `public int getSourceId()` and `public String getSourceDescription()` methods that are available in every kind of packages.

[4] Function available only in PRO version.

PERMISSIONS

SCCT for Android needs internet connection to work. For using it you have to add in the AndroidManifest the “android.permission.INTERNET” permission.

The Android Manifest should look like this:

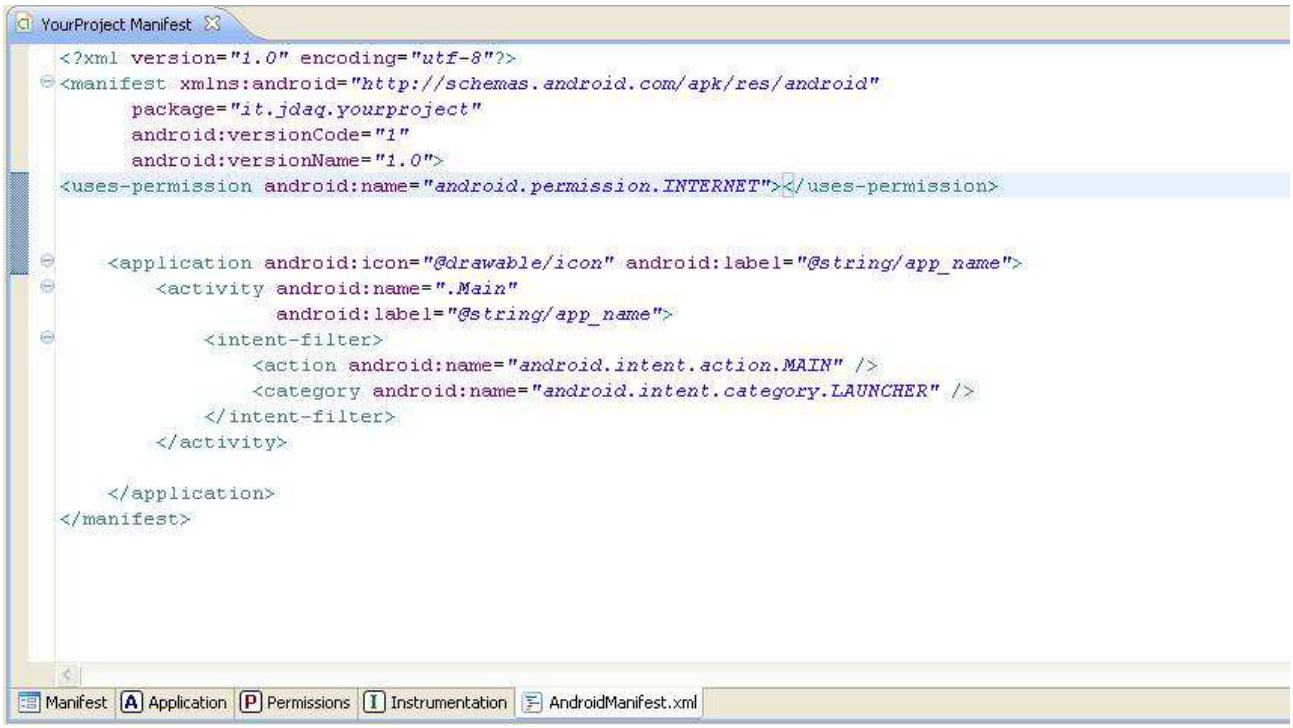


FIGURE 7 - ANDROIDMANIFEST

If you want to use saveFileOnExternalStorage() method of FilePackage class to save file on external storage you have to add also “android.permission.WRITE_EXTERNAL_STORAGE” permission.

QUICK START

To start, you need to create the project and include the library as showed in the **Installation** chapter and you have to add permission how showed in **Permission** chapter .

Then you can start to create the ConcreteObserver class, as inner class or normal class. In this example, the normal class is used.

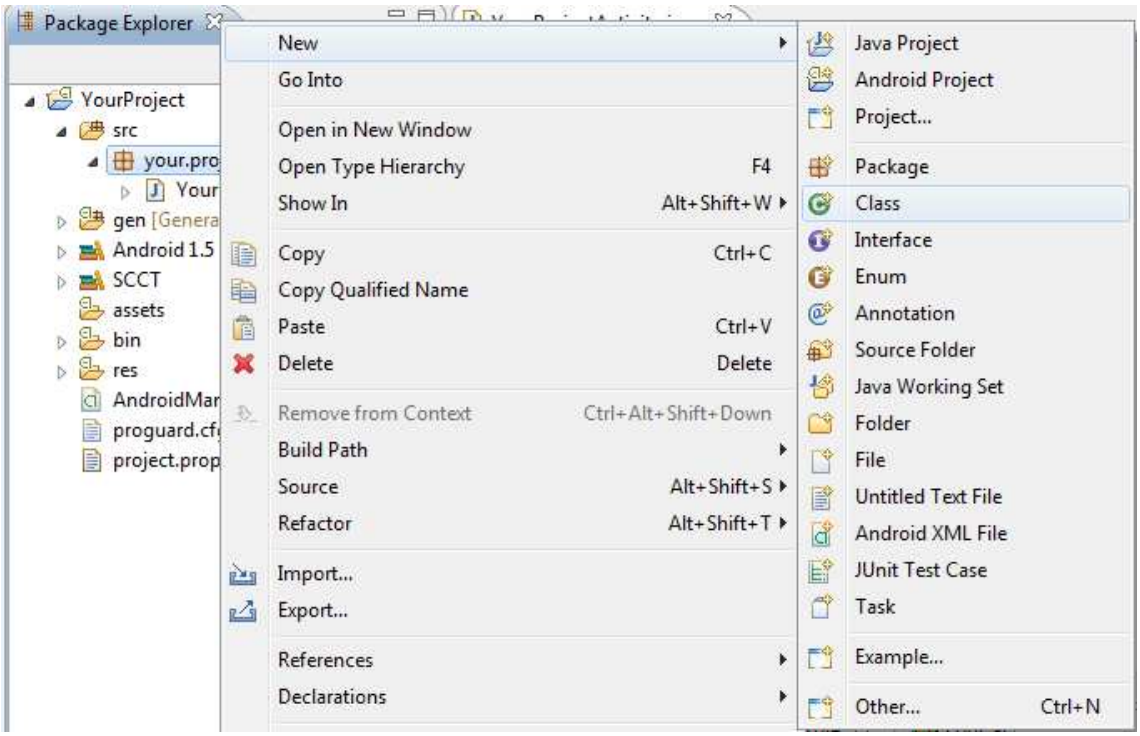


FIGURE 8 - NEW JAVA CLASS MENU ON ECLIPSE

Extend the ConcreteObserver with the SCCT_Observer class provided by the library.

```
public class ConcreteObserver extends SCCT_Observer {
```

At the beginning the IDE returns an error. That's because SCCT.JavaSubscriber.SCCT_Observer is not included. The editor should provide a hint. If doesn't happen, you can add it manually:

```
import SCCT.JavaSubscriber.SCCT_Observer;
```

Now you can override one or more listeners of `SCCT_Observer` class. In this example we implementing analogDataListener, digitalDataListener and configurationListeners:

```
@Override
public void analogDataListener(SCCT_AnalogDataPackage analogData){
      Log.d("SCCT",analogData.getCalendar().getTime().toString()
                  + " – analog data received");
}

@Override
public void digitalDataListener(SCCT_DigitalDataPackage digitalData){
      Log.d("SCCT",digitalData.getCalendar().getTime().toString()
                  + " – digital data received");
```

```
}

@Override
public void configurationListener(SCCT_ConfigurationPackage config){
      Log.d("SCCT",config.getCalendar().getTime().toString()
                  + " – configuration received");
}
```

The ConcreteObserver is ready. Let's see how implement the main activity class. Create an instance of SCCT_Subject class and another one of ConcreteObserver class.

```
SCCT_Subject subject = new SCCT_Subject();
ConcreteObserver observer = new ConcreteObserver();
```

Now you have to register the ConcreteObserver in the SCCT_ Subject object.

```
subject.registerObserver(observer);
```

Finally you have to open the communication and ask the data transmission's start.

```
try {
     subject.openCommunication("address",8083,"ApiKey","Description");
     subject.start();
} catch(IOException e) {
      //Handle exception
}
```

If the parameters of the openCommunication method are correct, you should see something similar to this screen in your logCat:

L...	Time	PID	Application	Tag	Text
D	06-08 17:11:5...	655	your.project	SCCT	Fri Jun 08 19:11:52 GMT+00:00 2012 - configuration received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:51 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:51 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:53 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:53 GMT+00:00 2012 - analog data received

FIGURE 9 - FINAL RESULT

FIGURE INDEX

FIGURE 1 - LIBRARY MANAGER WINDOW ON NETBEANS..... **ERRORE. IL SEGNALIBRO NON È DEFINITO.**

FIGURE 2 - PROJECT PROPERTIES WINDOW ON ECLIPSE.....10

FIGURE 3 - COMMUNICATION SCHEME 11

FIGURE 4 - UML SCHEME OF OBSERVER PATTERN 11

FIGURE 5 - HOW PACKAGES ARE SORTED IN QUEUES.....12

FIGURE 6 - ANALOG DATA MATRIX 14

FIGURE 7 - SCENARIO WITH A MULTIPLE DATA SOURCE SERVER AND CLIENTS THAT NEED A SUBSET OF PUBLISHED DATA. 24

FIGURE 8 - ANDROIDMANIFEST 25

FIGURE 9 - NEW JAVA CLASS MENU ON ECLIPSE 26

FIGURE 10 - FINAL RESULT.....27